Mining a World of Smart Sensors

Suman Nath Amol Deshpande, Phillip B. Gibbons, Srinivasan Seshan

IRP-TR-02-05
August 2002

Intel Research Pittsburgh

Mining a World of Smart Sensors

Suman Nath[†,*] Amol Deshpande[‡,*] Phillip B. Gibbons[*] Srinivasan Seshan[†,*]

[*]*Intel Research Pittsburgh* [†]*Carnegie Mellon University* [‡]*U.C. Berkeley*

Abstract

The proliferation and affordability of webcams has created opportunities for exciting new classes of distributed services. A key stumbling block to mining these rich information sources is the lack of a common, scalable networked infrastructure for collecting, filtering, and combining the video feeds, extracting the useful information, and enabling distributed queries. This paper describes the design of such an infrastructure, called *IRIS* (for *Internet-scale Resource-Intensive Sensor services*). IRIS is a potentially global network of smart sensor nodes, with webcams or other sensors, and organizing nodes that provide the means to query recent and historical sensor-based data. IRIS exploits the fact that high-volume sensor feeds are typically attached to devices with significant computing power and storage, and running a standard operating system. Aggressive filtering, smart query routing, and semantic caching are used to dramatically reduce network bandwidth utilization and improve query response times, as demonstrated by experiments with an IRIS prototype. We consider a number of potential sensor-based services enabled by IRIS, including a service for finding the cheapest available parking space near a user's destination.

1 Introduction

Imagine driving towards a destination in a busy metropolitan area. While stopped at a traffic light, you query your PDA specifying your destination and criteria for desirable parking spaces (e.g., within two blocks of your destination, at least a four hour meter). You get back directions to an available parking space satisfying your criteria. Hours later, you realize that your meter is about to run out. You query your PDA to discover that, historically, meter enforcers are not likely to pass by your car in the next hour. A half hour later, you return to your car and discover that although it has not been ticketed, it has been dented! Querying your PDA, you get back images showing how your car was dented and by whom.

This scenario demonstrates the potential utility of sensor-based services such as a Parking Space Finder, Silent (Accident) Witness and Meter Enforcement Tracker. While several research projects [5, 4, 2] have begun to explore the use of networked collections of sensors, these systems have targetted the use of closely co-located resource-constrained sensor "motes" [6, 9]. In this paper, we describe a sensor network system architecture, called *IRIS* (for *Internet-scale Resource-Intensive Sensor services*), based on much more intelligent participants. We envision an environment where different nodes on the Internet (standard PCs, laptops and PDAs) have attached sensors such as webcams (video cameras attached to Web-enabled devices). Any sensor-based service can retrieve information from this collection of sensors and provide service to users.

While webcams are inexpensive and easy to deploy across a wide area, realizing useful services requires addressing a number of challenges:

- Preventing the transfer of large data feeds across the network is necessary for system scalability.
- Efficiently discovering relevant data among the distributed collection of sensor and service nodes and delivering it to interested participants is crucial for reasonable response times.
- Efficiently handling static meta-data information (e.g., parking meter details and map directions), live readings from multiple sensor feeds, readings from the recent past, and long term historical data is required in order to answer queries like those in our example scenario.

Our goal in IRIS is to create a common, scalable software infrastructure that allows services to address these challenges in a manageable fashion. This would enable rapid development and deployment of distributed services over a worldwide network of sensor feeds.

IRIS is composed of a potentially global collection of Sensing Agents (SAs) and Organizing Agents (OAs). SAs collect and process data from their attached webcams or other sensors, while OAs provide facilities for

querying recent and historical sensor data. Any Internet connected, PC-class device can play the role of an OA. Less capable PDA-class devices can act as SAs. Continued advances in microprocessing technology enable significant processing power and memory to be encapsulated in smaller and cheaper devices that may act as SAs.[1] Key features of IRIS include:

- Providing simple APIs for orchestrating the SAs and OAs to collect, collaboratively process and archive sensor data while minimizing network data transfers.
- Handling issues of service discovery, query routing, semantic caching of responses and load balancing in a scalable manner for all services.
- Handling the demanding requirements of processing and querying live and historical high-volume sensor feeds such as webcams.

We believe that IRIS can enable a wealth of new sensor-based services. Example uses include providing live virtual tours of cities, answering queries about the waiting time at different restaurants, unobtrusive monitoring of your children playing in the neighborhood, witnessing whose dog pooped on your lawn, and determining where an umbrella was left behind.

The remainder of this paper is organized as follows. Section 2 gives an overview of the IRIS architecture. Section 3 presents initial experimental results. Section 4 compares IRIS with related work. Section 5 presents conclusions and future work.

2 The IRIS Architecture

In this section we describe IRIS, presenting its overall architecture, its query processing features, its caching and data consistency mechanisms, and its support for service development and deployment.

Architecture. IRIS is composed of a dynamic collection of SAs and OAs. Nodes in the Internet participate as hosts for SAs and OAs by downloading and running IRIS modules. Sensor-based services are deployed by orchestrating a group of OAs dedicated to the service. These OAs are responsible for collecting and organizing the sensor data in a fashion that allows for a particular class of queries to be answered (e.g., queries about parking spaces). The OAs index, archive, aggregate, mine and cache data from the SAs to build a system-wide distributed database for that service. Having separate OA groups for distinct services enables each service to tailor the database schema, caching policies, data consistency mechanisms, and hierarchical indexing to the particular service. This does not restrict the placement of OAs, because multiple OAs can be hosted on the same node.

In contrast, SAs are shared by all services. An SA collects raw sensor data from a number of (possibly different types of) sensors. The types of sensors can range from webcams and microphones to temperature and pressure gauges. The focus of our design is on sensors that produce large volumes of data and require sophisticated processing, such as webcams. SAs with attached webcams include, as part of the IRIS module, Intel's open-source image-processing library [1]. The sensor data is copied into a shared memory segment on the SA, for use by any number of sensor-based services.

OAs upload scripting code to any SA collecting sensor data of interest to the service, basically telling the SA to take its raw sensor feed, perform the specified processing steps, and send the distilled information to the OA. For video feeds, the script consists primarily of calls to the image processing library. Filtering data at the SAs prevents flooding the network with high bandwidth video feeds and is crucial to the scalability of the system. Even compressed video consumes considerable bandwidth, whereas aggressive filtering can reduce 10 minutes of video down to under a kilobyte of data, depending on the service. For example, the Parking Space Finder service distills the video down to a bit vector indicating which spots are empty.

One key concern is that because multiple services utilize each SA, the relatively limited computing power of the SA may be overloaded. Therefore, it is important that SAs are used efficiently and avoid repetitious work. Thus, our programming API for the SA allows downloaded code from different OAs to cooperate and avoid redundant processing and storage. For example, both the Parking Space Finder service and the Silent Accident Witness service perform common steps of filtering for motion detection and for identifying vehicles. Opportunities for work sharing can be detected automatically by the SA, by observing matching sequences of library calls on the incoming stream of frames. We anticipate significant work sharing, because the downloaded codes are all specialized to the processing of the SA's particular sensor feeds.

[1] Note that SAs (and OAs) need not have keyboards or displays.

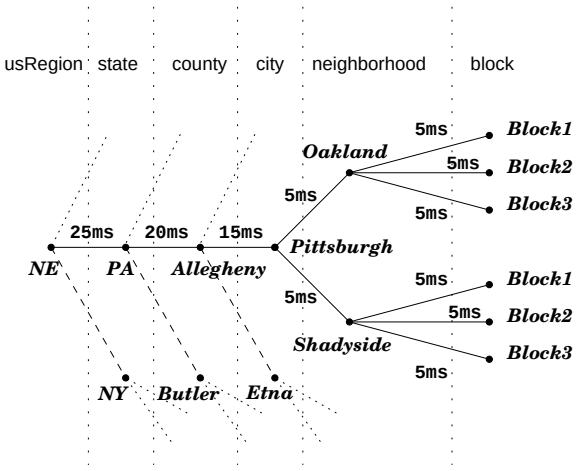

Figure 1: OA Hierarchy

Query Processing. Central to IRIS is distributed query processing. Data is stored in XML databases associated with each OA. We envision a rich and evolving set of data types, aggregate fields, etc., best captured by self-describing tags – hence XML was a natural choice. Larger objects such as video frames are stored outside the XML databases; this enables inter-service sharing, as well as more efficient image and query processing.

Data for a particular service is organized hierarchically, with each OA owning a part of the hierarchy. An OA may also cache data from one or more of its descendants. A common hierarchy for OAs is geographic, because each sensor feed is fundamentally tied to a particular geographic location.[2] Figure 1, for example, shows a geographical hierarchy for Parking Space Finder.

A user's query, represented in the XPATH language, selects data from a set of nodes in the hierarchy. The query in Figure 2, for example, selects data from the *Oakland Block1* and *Shadyside Block1* nodes in the hierarchy of Figure 1. We exploit the hierarchical nature of the OA organization to expedite the routing of queries to the data, as follows. Observe that each query contains a (maximal) hierarchical prefix, which specifies a single path from the root of the hierarchy to the lowest common ancestor (LCA) of the nodes potentially selected by the query. For the query in Figure 2, this is the prefix of the query up to `city[@id='Pittsburgh']`. We denote this LCA node (in our example, the *Pittsburgh* node) as the *starting point* for the query.

In IRIS, a query from a user anywhere in the world

[2] A service may define indices based on non-geographic hierarchies. In IRIS, such hierarchies are reflected in the XML schema.

```
/usRegion[@id='NE']/state[@id='PA']
/county[@id='Allegheny']/city[@id='Pittsburgh']
/neighborhood[@id='Oakland' OR @id='Shadyside']
/block[@id='1']/parkingSpace[available='yes']
```

Figure 2: Query asking for all available parking spaces in *Oakland Block1* or *Shadyside Block1*.

is first routed to its starting point. But how do we find the starting point OA, given the large number of OAs and the dynamic mapping of OAs to host machines? Our solution is to have DNS-style names for OAs that can be constructed from the queries themselves, to create a DNS server hierarchy identical to the OA hierarchy, and to use DNS lookups to determine the IP addresses of the desired OAs. For our example query, we construct the DNS-style name `pittsburgh.allegheny.pa.ne.parking.intel-iris.net`, perform a DNS lookup to get the IP address of the *Pittsburgh* OA, and route the query there.

Upon receiving a query, the starting point OA queries its local database and cache, and evaluates the result. If necessary, it gathers missing data by sending subqueries to its children OAs (the *Oakland* and *Shadyside* OAs in our example), who may recursively query their children, and so on. Finally the answers from the children are combined and the result is sent back to the user. Note that the children IP addresses are found using the same DNS-style approach, with most lookups served by the local host.

To handle unanticipated historical queries (e.g., show me images of my car being dented), a circular buffer of recent sensor data (e.g., video frames) is maintained at the SA in a compressed format.

Semantic Caching and Data Consistency. An OA may cache query result data from one or more of its descendants. Subsequent queries may use this cached data, even if the new query is not an exact match for the original query. For example. the query in Figure 2 may use data for *Oakland* cached at the *Pittsburgh* OA, even though this data is only a partial match for the new query. Similarly, if distinct *Oakland* and *Shadyside* queries result in the data being cached at *Pittsburgh*, the query may use the merged cached data to immediately return an answer. The current prototype supports a limited form of such *semantic caching*.

Due to delays in the network and the use of cached data, answers returned to users will not reflect the absolutely most recent data. Instead, queries specify a consistency criteria indicating a tolerance for stale data

(or other types of approximation). For example, when heading towards a destination, it suffices to have a general idea of parking space availability. However, when arriving near the destination, exact spaces are desired. We store timestamps along with the data, so that an XPATH query specifying a tolerance is automatically routed to the data of appropriate freshness. In particular, each query will take advantage of cached data only if the data is sufficiently fresh.

Service Development and Deployment. IRIS seeks to make it (relatively) easy to create and deploy new services. There are basically four steps to developing a new service using IRIS:

1. Locate the desired webcams and other sensors, using IRIS's service discovery mechanisms.
2. Create an XML schema, which defines the attributes and tags used to archive the data and the hierarchies used to index the data.
3. Download scripts to the SAs that define the push of new sensor data into the OA databases.
4. Create a web-based front end suitable for the service, which converts form-based user queries into XPATH queries.

IRIS will seamlessly handle query processing, indexing, networking, caching, load balancing, and resource sharing, on behalf of all the services. This is in addition to the significant advantage provided by IRIS's unified software platform.

3 Experimental Results

We have implemented an early prototype of IRIS, incorporating live feeds from webcams operating in a controlled environment, which are processed in the SAs as directed by scripts downloaded from the OAs. User queries are answered by the OAs following the steps described in the previous section. SAs are written in C and OAs are written in Java.

This section presents an experimental study of our prototype, which seeks to answer the following three questions: (1) What is the raw performance of the prototype on simple queries, in terms of processing time, communication time and querying time? (2) What are the performance gains in leveraging the intelligence at the SAs vs. performing the work at the OAs? (3) How critical is accurate routing of queries and semantic caching to system performance?

Method	Bandwidth (bps)
Raw camera feed (30 FPS)	221184000
1 FPS sampling	7372800
Compressing in SA (1 FPS)	51200
Filtering in SA (1 FPS)	256

Table 1: Bandwidth requirements of the data sent from the SA to the OA under four scenarios.

Experimental Setup. In our experiments, we run SAs on 550 MHz Pentium III computers with 128 MB RAM and OAs on 1.6GHz Pentium IV computers with 512 MB RAM. All the machines run Redhat 7.3 Linux with kernel 2.4.18. We study a Parking Space Finder service that uses the hierarchy shown with solid lines in Figure 1. SAs get video feeds from cameras taking pictures of parking lots at a rate of 30 frames per second.

We use two machines to run the OAs, where the first hosts the *NE*, *Allegheny*, *Oakland*, and *Shadyside Blocks* OAs and the second hosts the *PA*, *Pittsburgh*, *Shadyside*, and *Oakland Blocks* OAs. For the queries in our study, this configuration ensures that all communication is between OAs on different machines and at most one OA per machine will be performing work at any time. To simulate wide area network latencies, we add (realistic) artificial delays on the links between OAs, as shown in Figure 1.

Processing Webcam Feeds. In the first experiment, we download to the SAs filter code that processes one video frame per second, extracts the current states of the parking spots and sends the information back to the subscribing OAs, who update their local databases. Because OAs are more powerful than SAs, an alternative is to perform the image filtering in the OAs: SAs send compressed video frames to the OAs, who then decode the frames, process them with the filter code, and update their local databases. We use the *FAME* library for encoding the video frames into MPEG format in SAs, and the *SMPEG* library for decoding the frames in OAs.

Table 1 shows how placing the filter code in the SAs reduces the required bandwidth between SAs and OAs. Although cameras feed a large volume of raw video data to the SAs[3], our Parking Space Finder service samples the frames at only 1 frame per second. Still, sending these uncompressed frames to the OAs demands a huge amount of bandwidth. As we can see from the table, encoding the frames in MPEG format significantly

[3]Most webcams compress the video to less than 12Mbps to transfer it across a USB bus.

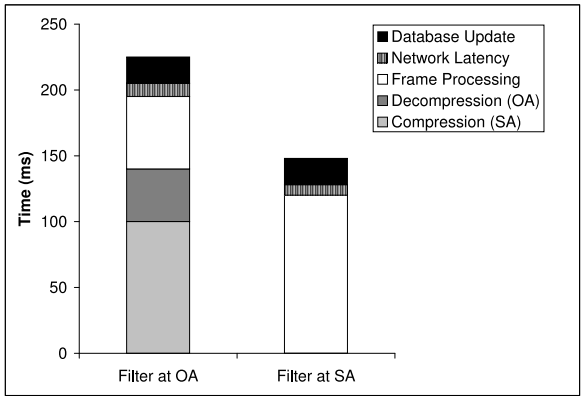

Figure 3: Time spent on various stages to extract information from video frames and update the database.

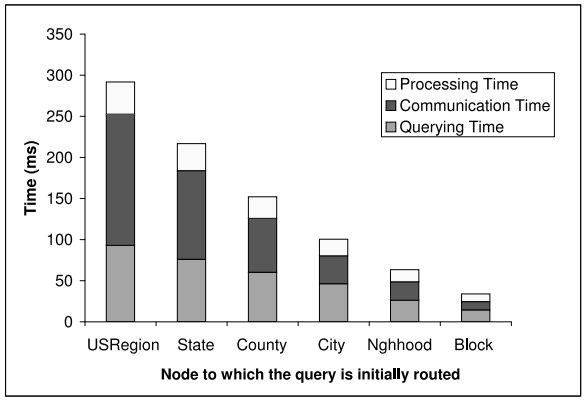

Figure 4: Time taken by a query depending on where it is routed initially.

reduces the traffic (almost 144 times for our service). However, filtering the frames in the SAs produces the least volume of traffic, as little as a few bytes per frame.

Figure 3 shows the amount of time spent on various stages to extract useful information from a video frame and update the database, under the two alternatives. In both scenarios, the network latency and database update times are a relatively small component of the overall time. Filtering at the OAs reduces the image processing time by a factor of two, but the overall time is 50% slower because of the overheads for MPEG encoding and decoding.

Query Execution. In our next experiment, we show the break-down of the end-to-end time of a query as a function of the node in the hierarchy at which it is routed to initially. We consider a simplified version of our earlier example query (Figure 2). This simplified query requests all information on a single parking space in *Oakland*. Figure 4 shows the time spent in three main components: (1) the network communication time, (2) time spent in querying the database at the nodes, and (3) the processing time (essentially the rest of the time).

As we can see, if the query is initially routed to a higher level node, most of the time is spent in network communication because of the high latencies between the upper-level nodes. As we come down to the lower-level nodes, the time taken to query the databases dominates the end-to-end query time. This also demonstrates the importance of intelligently routing a query directly to the starting point node, as is done by IRIS This particular query will be routed immediately to the block-level OA and most of the network hops (and database accesses) are avoided.

Our final experiment demonstrates the effect of data

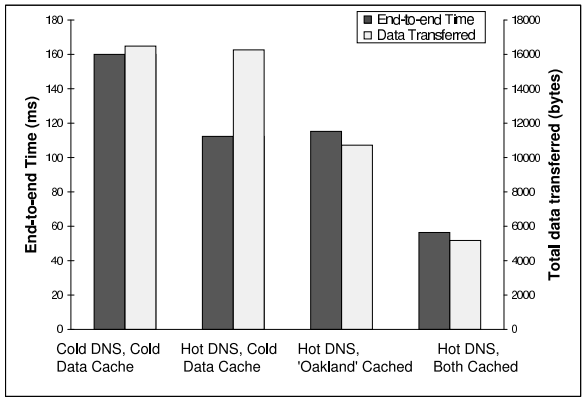

Figure 5: Effect of DNS and data caches on end-to-end query time and total data transferred.

and DNS caching on query response times. Figure 5 shows the end-to-end time and the total data transferred during its execution under various scenarios. The query studied requests all information on one parking space in each of *Oakland* and *Shadyside*. The starting point for this query is the *Pittsburgh* node. The first two scenarios show that having the starting node's DNS entry cached at the node at which the query is posed decreases the query response time by over 25%. Next, we cache part of the answer at the *Pittsburgh* node, namely the data corresponding to the *Oakland* node. Though the amount of network communication goes down by about 30%, the response time is basically unchanged because the time is dominated by the *Pittsburgh-Shadyside-Block1* path. But we see a significant decrease in the response time (more than 50%) when data corresponding to both the *Oakland* and *Shadyside* nodes is cached. This illustrates the significant advantage of (semantic) caching in the network.

4 Related Work

Due to the emphasis of past efforts on resource-constrained motes, earlier key contributions have been in the areas of tiny operating systems [8] and low-power network protocols [10]. Mote-based systems have relied on techniques such as directed diffusion [7] to direct sensor readings to interested parties or long-running queries [3] to retrieve the needed sensor data to a front-end database. Other groups have explored using query techniques for streaming data and using sensor proxies to coordinate queries [11] to address the limitations of sensor motes. None of this work considers sensor networks with intelligent sensor nodes, high-bandwidth sensors, and worldwide scale.

The distributed database infrastructure in IRIS shares much in common with a variety of large-scale distributed databases. For example, DNS [12] relies on a distributed database that uses a hierarchy based on the structure of host names, in order to support name-to-address mapping. LDAP [13] addresses some of DNS's limitations by enabling richer standardized naming using hierarchically-organized values and attributes. However, it still supports only a relatively restrictive querying model.

5 Conclusions and Future Work

IRIS is a software infrastructure for mining a world of webcams and other sensors. We have discussed features in IRIS that greatly simplify the tasks of collecting, filtering, and combining sensor feeds, extracting the useful information, and enabling distributed queries with reasonable response times. Query times are significantly reduced through our novel marriage of hierarchical schemas, OA hierachies, DNS-style names extracted from the hierarchical prefixes of queries, and semantic caching within the hierarchy. We have reported preliminary performance numbers demonstrating the efficacy of our approach.

The work on IRIS is ongoing. The next steps we are actively taking include investigating additional services, making the architecture more robust by handling both failures and graceful departures of nodes, improving the performance through smarter semantic caching, and providing mechanisms for new SAs to be incorporated into running services. The latter requires a service discovery system supporting continuous, geographically-scoped queries. Other future work will explore privacy and security issues, scheduling / protection / resource allocation for the executing code, additional data consistency models, and adapting to changes in operating conditions.

We envision IRIS as an enabling technology for extracting useful information from the vast sensor feeds and presenting it to users in a timely manner. IRIS exploits the fact that high-volume sensor feeds are typically attached to devices with significant computing power and storage, and running a standard operating system. By providing an architecture that makes it easy to dynamically download filtering code to such devices, IRIS achieves both flexibility and scalability.

Acknowledgements. We thank M. Satyanarayanan for many valuable suggestions and C. Helfrich for helping with the experimental setup.

References

[1] Intel Open Source Computer Vision Library. http://www.intel.com/research/mrl/research/opencv/.

[2] Webdust: Automated construction and maintenance of spatially constrained information in pervasive microsensor networks. http://athos.rutgers.edu/dataman/webdust.

[3] BONNET, P., GEHRKE, J. E., AND SESHADRI, P. Towards sensor database systems. In *MDM* (2001).

[4] CULLER, D., BREWER, E., AND WAGNER, D. Berkeley Wireless Embedded Systems (WEBS). http://webs.cs.berkeley.edu/.

[5] ESTRIN, D., GOVINDAN, R., AND HEIDEMANN, J. SCADDS: Scalable Coordination Architectures for Deeply Distributed Systems. http://www.isi.edu/scadds.

[6] ESTRIN, D., GOVINDAN, R., HEIDEMANN, J., AND KUMAR, S. Next century challenges: Scalable coordination in sensor networks. In *MOBICOM* (1999).

[7] HEIDEMANN, J., ET AL. Building efficient wireless sensor networks with low-level naming. In *SOSP* (2001).

[8] HILL, J., ET AL. System architecture directions for network sensors. In *ASPLOS* (2000).

[9] KAHN, J., KATZ, R. H., AND PISTER, K. Next century challenges: Mobile networking for 'smart dust'. In *MOBICOM* (1999).

[10] KULIK, J., RABINER, W., AND BALAKRISHNAN, H. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks . In *MOBICOM* (1999).

[11] MADDEN, S., AND FRANKLIN, M. J. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE* (2002).

[12] MOCKAPETRIS, P. V., AND DUNLAP, K. J. Development of the Domain Name System. In *SIGCOMM* (1988).

[13] WAHL, M., HOWES, T., AND KILLE, S. Lightweight Directory Access Protocol (v3). Tech. rep., IETF, 1997. RFC 2251.